Linting Rules

Play+ Linting Guide : playlint Introduction ■ In the Play+ ecosystem, we believe the codebase is a user interface for developers. It must be as intuitive, clean, and predictable as the products we ship. This guide is based on the concept of codified quality standards . Our linting setup is essential for creating a frictionless and efficient development experience by automatically enforcing a consistent style and flagging potential issues early. This directly supports our core design pillars: making our codebase Distinct through uniform style, Intuitive by clarifying intent and reducing cognitive load, and Inclusive by baking in accessibility checks from the first line of code. Package Info ■ The Play+ linting configuration is provided as a development dependency and is pre-installed in all Golden Path starter kits. Description Package / Path Golden Path (Recommended) Pre-installed as a devDependency Uplift Path npm install --save-dev @playplus/eslint-config Folder Reference ■ The linting setup relies on a few key configuration files at the root of your project. File / Directory Purpose & Guidelines .eslintrc.js The main ESLint configuration file. It extends the base @playplus/eslint-config. .prettierrc Configuration file for Prettier, ensuring consistent code formatting. package.json Defines the playlint script and lists the dev dependencies. reports/ The git-ignored directory where automated linting reports are saved. Helper - Pillars Alignment ■ Our automated linting strategy is a direct implementation of our core design pillars, applied to the code itself. Pillar How This Helper Aligns Distinct Enforces a uniform coding style and naming convention, creating a recognizable brand identity at the code level. Intuitive Codifies best practices to reduce cognitive load, making code easier to read, reason about, and maintain. Inclusive Integrates accessibility (a11y) checks directly into the linter, ensuring products are built for everyone from the start. Helper Overview ■ Unlike other helpers that provide runtime APIs, the Play+ linting setup is a zero-configuration toolchain . It's delivered as an ESLint configuration package (@playplus/eslint-config) and a set of pre-configured scripts. Its purpose is to abstract the plumbing of setting up and maintaining a modern linting environment. It automates quality control across the development lifecycle: During Development : Provides real-time feedback in your IDE. On Commit : A pre-commit hook automatically checks and fixes staged files, blocking critical issues. In Continuous Integration : A playlint script runs on every pull request, failing the build if errors are present to protect the main branch. This system ensures that every developer on every

team adheres to the same high standards of quality, security, and accessibility without needing to configure anything themselves. Config Options ■ While the setup is designed to be zero-config, you can override specific rules in your project's .eslintrc.js file. This should be done sparingly and with team consensus. Config File Action Example .eslintrc.js Override a rule rules: { 'no-console': ['off', { allow: ['warn', 'error'] }] } .eslintrc.js Extend configuration extends: ['@playplus/eslint-config', 'plugin:my-plugin/recommended'] Example .eslintrc.js: // .eslintrc.js module . exports = { extends : ["@playplus/eslint-config"] , rules : { // Overrides the default rule to allow console.warn and console.error "no-console" : ["error" , { allow : ["warn" , "error"] }] , } , } ; Key Scripts & Commands ■ The functionality is exposed through npm scripts defined in your package.json. Script Name What It Does Default Command playlint Runs the ESLint checker across all relevant source files in the project. eslint . playlint :fix Runs ESLint with auto-fix enabled to automatically resolve fixable issues. eslint . --fix playlint :report Runs the linter and generates a machine-readable JSON report in the reports/ directory. eslint . --format json --output-file reports/lint-report.json playlint:report :html Generates a comprehensive HTML report with visual analytics and filtering. Custom script that processes JSON and creates HTML format Formats all code using Prettier. prettier --write . format :check Checks formatting without making changes. prettier --check . React & Angular: Automated Enforcement ■ For both frameworks, the Golden Path provides complete automation. There is nothing to set up. In your IDE : With the recommended extensions (ESLint/Prettier), you get real-time feedback as you type. When you commit : A pre-commit hook (via Husky) automatically lints your changes. When you create a Pull Request : A GitHub Action runs npm run playlint , ensuring no errors can be merged. VS Code IDE Setup ■ To get the best real-time experience, create a .vscode/settings.json file with the following content: { "editor.formatOnSave" : true , "editor.codeActionsOnSave" : { "source.fixAll.eslint" : true } , "eslint.validate" : ["javascript" , "javascriptreact" , "typescript" , "typescriptreact"] , "editor.defaultFormatter" : "esbenp.prettier-vscode" } Live Dashboard Component ■ The linting report dashboard is available as a component in your application: import { LintingReportComponent } from "./components/linting-report/linting-report.component" ; // Use in your template < app - linting - report > < / app - linting - report > ; Static HTML Reports ■ Generate static HTML reports using npm scripts: # Generate linting report with HTML output npm run playlint:report:html # This will: # 1. Run ESLint and save JSON output to reports/linting-report.json # 2. Generate beautiful HTML report at reports/linting-report.html

Additional Info ■ Why We Built This ■ Configuring a modern linting toolchain is complex. It involves selecting and integrating multiple tools (ESLint, Prettier), plugins (for React, Angular, a11y, security), and defining hundreds of rules. Without a centralized solution, each team would waste time on setup and debates, leading to inconsistencies across projects. The @playplus/eslint-config package solves this by providing a single, opinionated, and production-ready configuration. It eliminates boilerplate and configuration drift , ensuring every project starts with and maintains the same high-quality bar. Best Practices ■ Trust the Automation : Let the pre-commit hooks and CI checks do their job. Use Disables Sparingly : Only use // eslint-disable-next-line for true edge cases and always add a comment explaining why it's necessary. Integrate Your IDE : A properly configured IDE gives you the fastest feedback loop. Review Reports : Periodically check the generated lint-report.json to identify recurring patterns or areas for team-wide improvement. Developer Checklist ■ Have I installed the recommended ESLint and Prettier extensions for my IDE? Is my IDE configured for format-on-save and fix-on-save? When I need to disable a rule, have I added a clear, explanatory comment? Am I letting the CI/CD pipeline validate my code quality before merging? Linting Standards & Rule Coverage ■ The @playplus/eslint-config package enforces a comprehensive set of rules by default. The following standards are automatically applied to all projects to ensure consistency, quality, and safety. While these rules can be toggled by developers in their local configuration, it is not advised as it creates divergence from the Play+ standard. Code Quality ■ Key (Rule) Default Value Description prefer-const error Requires const declarations for variables that are never reassigned. no-var error Disallows the use of var in favor of let and const. prefer-arrow-callback error Enforces the use of arrow functions for callbacks. no-unused-vars warn Flags variables that are declared but never used. complexity ['error', 10] Limits cyclomatic complexity to prevent overly complex functions. import/order error Enforces a consistent and logical order for import statements. eqeqeq error Requires the use of strict equality operators === and !==. prefer-template error Enforces the use of template literals instead of string concatenation. no-throw-literal error Restricts what can be thrown as an exception, requiring Error objects. no-eval error Disallows the use of the eval() function to prevent security risks. max-depth ['warn', 4] Warns when code is nested too deeply. unused-imports/no-unused-imports warn Flags unused import statements that can be safely removed. Accessibility ■ Key (Rule) Default Value Description jsx-a11y/alt-text error Enforces that all <img> elements have meaningful alt text. jsx-a11y/label-has-associated-control error Requires that every form label is

associated with a form control. jsx-a11y/aria-props error Enforces that only valid ARIA props are used. jsx-a11y/click-events-have-key-events error Requires a keyboard event handler for elements with a click event. jsx-a11y/no-redundant-roles error Prevents the use of ARIA roles on elements that have implicit roles. Security ■ Key (Rule) Default Value Description react/no-danger warn Warns against the use of dangerouslySetInnerHTML. react/jsx-no-target-blank error Enforces rel="noreferrer" on links with target="_blank". security/detect-unsafe-regex warn Detects regular expressions that are susceptible to ReDoS attacks. security/detect-child-process warn Flags the use of child_process which can be a security risk. Framework: React ■ Key (Rule) Default Value Description react-hooks/rules-of-hooks error Enforces the Rules of Hooks to prevent common mistakes. react-hooks/exhaustive-deps warn Verifies dependency arrays in hooks like useEffect and useCallback. react/no-direct-mutation-state error Prevents direct mutation of this.state; setState must be used. react/jsx-key error Requires a unique key prop for elements in an array or iterator. Framework: Angular ■ Key (Rule) Default Value Description @angular-eslint/component-selector error Enforces a consistent prefix for component selectors (e.g., app-). @angular-eslint/no-empty-lifecycle-method warn Flags empty lifecycle methods that can be removed. @angular-eslint/template/accessibility-alt-text error Enforces alt text on <img> elements inside Angular templates. Formatting ■ Key (Rule) Default Value Description prettier/prettier error Runs Prettier as an ESLint rule and reports differences as errors. Report Features ■ Modern UI Dashboard ■ Beautiful, responsive design with gradient headers Interactive filtering by severity, category, and search Real-time statistics and charts Auto-fix functionality for fixable issues Export capabilities Comprehensive Statistics ■ Total issues count Breakdown by severity (errors, warnings, info) Auto-fixable issues count Category-based analysis Top rule violations Advanced Filtering ■ Filter by severity level Filter by category (security, quality, style, accessibility, performance) Search across files, rules, and messages Sort by multiple criteria Visual Analytics ■ Bar charts for category distribution Top rules analysis Interactive charts with hover effects Color-coded severity indicators Integration ■ With CI/CD ■ Add to your CI pipeline: # .github/workflows/lint.yml - name : Generate Linting Report run : npm run playlint : report : html - name : Upload Report uses : actions/upload - artifact@v2 with : name : linting - report path : reports/linting - report.html With IDE ■ Configure your IDE to use the generated reports: // .vscode/settings.json { "eslint.reportUnusedDisableDirectives" : "error" , "eslint.format.enable" : true , "eslint.autoFixOnSave" : true } Troubleshooting ■

Common Issues ■ Import ordering errors Run npm run playlint:fix to auto-organize imports Prettier conflicts Run npm run format to fix formatting issues Unused imports Run npm run playlint:fix to remove unused imports Complexity warnings Break down complex functions into smaller ones Rule Overrides ■ If you need to override a rule (use sparingly): { "rules" : { "no-console" : ["off" , { "allow" : ["warn" , "error" , "log"] }] } } Monitoring & Metrics ■ Key Metrics to Track ■ Number of linting errors per PR Time to fix linting issues Most common rule violations Auto-fix success rate Report Analysis ■ # Generate detailed report npm run playlint:report # Analyze report cat reports/lint-report.json | jq '.[] | select(.errorCount > 0) | .filePath'

```
// .eslintrc.js
module.exports = {
  extends: ["@playplus/eslint-config"],
  rules: {
    // Overrides the default rule to allow console.warn and console.error
    "no-console": ["error", { allow: ["warn", "error"] }],
  },
};


---

{
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "eslint.validate": [
    "javascript",
    "javascriptreact",
    "typescript",
    "typescriptreact"
  ],
  "editor.defaultFormatter": "esbenp.prettier-vscode"
}


---

import { LintingReportComponent } from "./components/linting-report/linting-report.component";

// Use in your template
<app-linting-report></app-linting-report>;


---

# Generate linting report with HTML output
```

```
npm run playlint:report:html

# This will:
# 1. Run ESLint and save JSON output to reports/linting-report.json
# 2. Generate beautiful HTML report at reports/linting-report.html


---

# .github/workflows/lint.yml
- name: Generate Linting Report
  run: npm run playlint:report:html
- name: Upload Report
  uses: actions/upload-artifact@v2
  with:
    name: linting-report
    path: reports/linting-report.html


---

// .vscode/settings.json
{
  "eslint.reportUnusedDisableDirectives": "error",
  "eslint.format.enable": true,
  "eslint.autoFixOnSave": true
}


---

{
  "rules": {
    "no-console": ["off", { "allow": ["warn", "error", "log"] }]
  }
}


---

# Generate detailed report
npm run playlint:report

# Analyze report
cat reports/lint-report.json | jq '.[] | select(.errorCount > 0) | .filePath'
```